Bioinformatics, 37(14), 2021, 2064–2065
doi: 10.1093/bioinformatics/btaa925
Advance Access Publication Date: 2 November 2020
Applications Note

OXFORD

Structural bioinformatics

quicksom: Self-Organizing Maps on GPUs for clustering of molecular dynamics trajectories

Vincent Mallet [1,2,†], **Michael Nilges**[1] **and Guillaume Bouvier** [1,*,†]

[1]Structural Bioinformatics Unit, Department of Structural Biology and Chemistry, Institut Pasteur, CNRS UMR3528, C3BI, USR3756, Paris 75015, France and [2]Mines ParisTech, Paris-Sciences-et-Lettres Research University, Center for Computational Biology, Paris 75272, France

*To whom correspondence should be addressed.
†The authors wish it to be known that these authors contributed equally.

Associate Editor: Arne Elofsson

Received on September 24, 2020; revised on October 14, 2020; editorial decision on October 15, 2020; accepted on October 20, 2020

Abstract

Summary: We implemented the Self-Organizing Maps algorithm running efficiently on GPUs, and also provide several clustering methods of the resulting maps. We provide scripts and a use case to cluster macro-molecular conformations generated by molecular dynamics simulations.
Availability and implementation: The method is available on GitHub and distributed as a pip package.
Contact: guillaume.bouvier@pasteur.fr

1 Introduction

We proposed in a former paper (Bouvier *et al.*, 2015) a Self-Organizing Map (SOM)-based algorithm to cluster macro-molecular conformations generated by molecular dynamics (MD) simulations. Alternative methods exist but they either rely on pairwise distance computation (González-Alemán *et al.*, 2020) or on including additional prior information (Olsson and Noé, 2019) whereas SOMs are simple linear clustering algorithms. Due to the expansion of the usage of graphics processing units (GPUs) to perform MD, the number of conformations from trajectories that need to be analyzed exploded. Therefore, our previous, CPU based, implementation of the SOM reached its limit. In the current paper, we propose a fast and efficient GPU implementation of SOM, `quicksom`. This is highly useful for the analysis of long MD trajectories, but can also be used for the clustering of other massive and high dimensional data.

In addition, we added a set of clustering tools that can be used on the maps produced by the methods. These tools serve to further summarize the inputs. They rely on either automatic clustering or a manual tool with a graphical interface. The efficiency of our tool is demonstrated through a case study on a long MD trajectory.

2 Efficient SOMs on GPU

Our implementation of SOMs (Kohonen, 1982) is based on PyTorch (Paszke et al., 2019). This alone speeds up operations and in addition allows us to use GPUs to make the computations even faster. We timed our method against the former application note (Bouvier *et al.*, 2015) and SOMPY(Moosavi et al., 2014), the main implementation of SOMs in Python that is a parallel CPU-based tool.

We used our method on some 2-dimensional toy data as well as on a MD trajectory resulting in 168-dimensional vectors. We show the time necessary to perform the training loop on 100k of these vectors in Table 1. We include a run on CPU for comparison as well as a run on several cores for SOMPY.

As expected, the new method is much faster than the old one, especially when run on the GPU, with a 160-fold speedup. We have comparable run times to the SOMPY implementation for the synthetic dataset, and twofold speedup on the higher dimensional MD data. This was unexpectedly fast for this CPU-based method. However, the SOMPY implementation does not support custom batch sizes, so the whole dataset is passed at once, which does not allow flexible training and biases the training for large datasets. We believe this to be a major limitation because tuning the optimization for large dataset was revealed to be key, in particular for the analysis of MD trajectories.

3 Clustering

We introduce several tools to cluster our data beyond a simple SOM cell affectation. The idea is to merge neighboring cells that represent several centroids inside the same cluster.

3.1 Automated approaches

The SOM yields a lattice graph whose nodes are the centroids and whose connectivity is given by the SOM grid. Edges are weighted by the Euclidean distance of the centroids they connect. We then compute the matrix of graph distances of nodes. To detect nodes that

Table 1. Mean time necessary to process 100k points in the training loop

Method	Toy data	MD trajectory
Old method (Bouvier *et al.*, 2015)	87 s	536 s
SOMPY (Moosavi *et al.*, 2014)	5.9 s	6.9 s
SOMPY (20 cores)	0.85 s	12.4 s
quicksom (CPU)	13 s	101 s
quicksom	1.6 s	3.3 s

belong to a given community and to cluste0r these nodes, we compute the topological distance matrix, and we can thus use algorithms such as the Agglomerative Clustering algorithm. The U-matrix is an informative 2D representation of the SOM that can depict efficiently its topology. A U-Matrix is defined as the matrix whose value at each grid point is the mean of the weight of its edges. To deal with toric connectivity, we re-arrange the map by flattening it following the graph shortest path from the global minimum, and pad it to get a square matrix. Using this formalism, we can turn our SOM into an image and use any segmentation algorithm to group centroids together. We default to the graph approach but implemented other algorithms that the user can choose.

3.2 Manual approaches

These automatic methods are always prone to failure because of the variety of possible maps resulting from the variety of possible data at hand as well as the choice of hyperparameters for the maps. Therefore, we also include a manual clustering option with a GUI, available as a command-line tool. The user can click on a point on the map and expand a region around it by hand, to select the relevant zones of the map they produced. We believe that this user-defined and application-specific clustering is a good work-around in case automatic clustering fails.

4 Molecular dynamics clustering

The tool can be applied to efficiently cluster MD trajectories. This is useful to create a library of representative structures. To do so, we represent each frame by the concatenation of each atom's coordinates and train a SOM to cluster these frames. We included a script to take a MD trajectory in the CHARMM (Brooks *et al.*, 2009) **dcd** format as input and output a **npy** file that can be handled by our SOM implementation. This script also allows the user to select a set of atoms of interest for the SOM analysis. We also included utilities to select the frames that fall into a given cluster for visualization.

The method was applied on the trajectory analyzed in our previous implementation (Bouvier *et al.*, 2015): 15 μs MD at 330 K of a simplified sequence of a 56-residue α/β subdomain of the protein G (Guarnera *et al.*, 2009) starting from an extended conformation. The analysis was performed on the C-α coordinates yielding 750 000 vectors of dimension 168. We include the results in Figure 1. We can see that the unfolded protein conformations correspond to a sparse region of the map while the most stable scaffolds fall into dense ones. We also included a representation of the transition steps as a flow map. This flow goes from the least populated and stable states to the more stable ones. It also enables visualization of the paths preferentially followed by the trajectory.

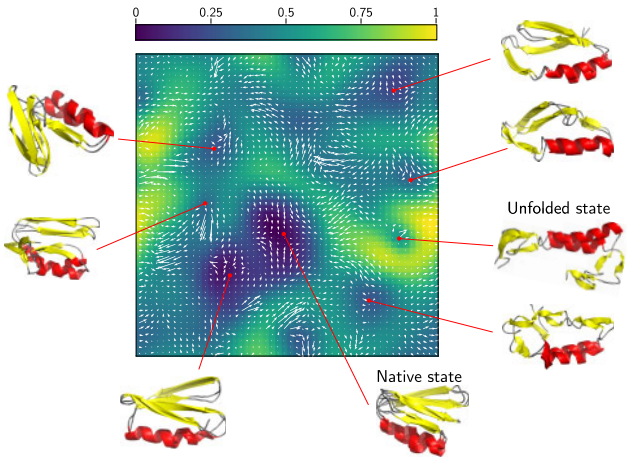

Fig. 1. Resulting U-Matrix of the clustering of a MD trajectory. The range of values is normalized. Darker color implies closer cells that represent a data cluster. White arrows represent the flow defined as the sum of the transition steps of the MD from each cell. Some structures were represented with a pointer to the cell they are mapped to by the algorithm

5 Implementation and availability

We have packaged our project into a pip package, `quicksom`, for easy setup and command line usage. The source code is available at https://github.com/bougui505/quicksom. A full description for installation and usage is available as a README.

Acknowledgements

The authors are grateful to the Institut Pasteur and the CNRS for their continued support for our research. The authors thank Laura Ortega and Mathias Ferber for feedback.

Financial Support: V.M. is recipient of a doctoral fellowship from the INCEPTION project [PIA/ANR-16-CONV-0005] and benefits from support from the CRI through Ecole Doctorale FIRE - Programme Bettencourt.

Conflict of Interest: none declared.

References

Bouvier,G. *et al.* (2015) An automatic tool to analyze and cluster macromolecular conformations based on self-organizing maps. *Bioinformatics*, **31**, 1490–1492.

Brooks,B.R. *et al.* (2009) CHARMM: the biomolecular simulation program. *J. Comput. Chem.*, **30**, 1545–1614.

González-Alemán,R. *et al.* (2020) BitClust: fast geometrical clustering of long molecular dynamics simulations. *J. Chem. Inf. Model.*, **60**, 444–448.

Guarnera,E. *et al.* (2009) How does a simplified-sequence protein fold? *Biophys. J.*, **97**, 1737–1746.

Kohonen,T. (1982) Self-organized formation of topologically correct feature maps. *Biol. Cybern.*, **43**, 59–69.

Moosavi,V. *et al.* (2014) *SOMPY: a python library for self organizing map (SOM). GitHub*. Accessed date: 2 Nov. 2020. https://github.com/sevamoo/SOMPY.

Olsson,S. and Noé,F. (2019) Dynamic graphical models of molecular kinetics. *Proc. Natl. Acad. Sci. USA*, **116**, 15001–15006.

Paszke,A. *et al.* (2019) Pytorch: an imperative style, high-performance deep learning library. In: Wallach,H. *et al.* (eds.) *Advances in Neural Information Processing Systems*, **Vol. 32**. Curran Associates, Inc., Massachusetts Institute of Technology, Cambridge, Massachusetts. pp. 8024–8035.